

Pemrograman Berorientasi Obyek

Polimorfisme dan
Binding

Polymorphism

- Polymorphism = Poly + Morphos
- Poly = banyak, Morphos = bentuk

Tujuan Polimorfisme

- Agar **komunikasi** antar obyek satu dengan lainnya dapat terjadi secara dinamis, dimana “message”/permintaan yang disampaikan dari satu obyek ke obyek lain bisa berbeda-beda tanggapannya dengan obyek lainnya.
- Diimplementasikan dalam respon yang berbeda-beda, melalui **pewarisan** dan implementasi **interface**
- Digunakan juga untuk standarisasi

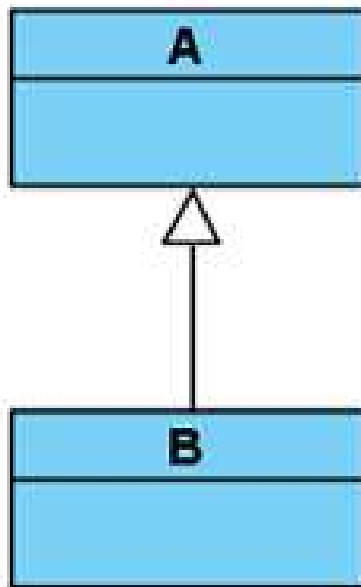
Polimorfisme

- Kemampuan obyek-obyek yang **berbeda** untuk memberi respon terhadap permintaan yang **sama**, *sesuai dengan cara masing-masing obyek*
 - Polymorfisme melalui inheritance (extends)
 - Polymorfisme melalui interface (implements)
- Jadi, Polymorfisme dapat diterapkan secara:
 - Overriding Method
 - Overloading Method

Polimorfisme

- Object **dinamis** suatu class induk dapat berperilaku seperti class turunan.
- Ketika object menunjuk class **induk**, object berperilaku seperti class **induk**.
- Ketika object menunjuk class **turunan**, object tersebut berperilaku seperti class **turunan**

Polimorfisme



`A contoh = new A();`

`B contoh = new B();`

`A contoh = new B();`

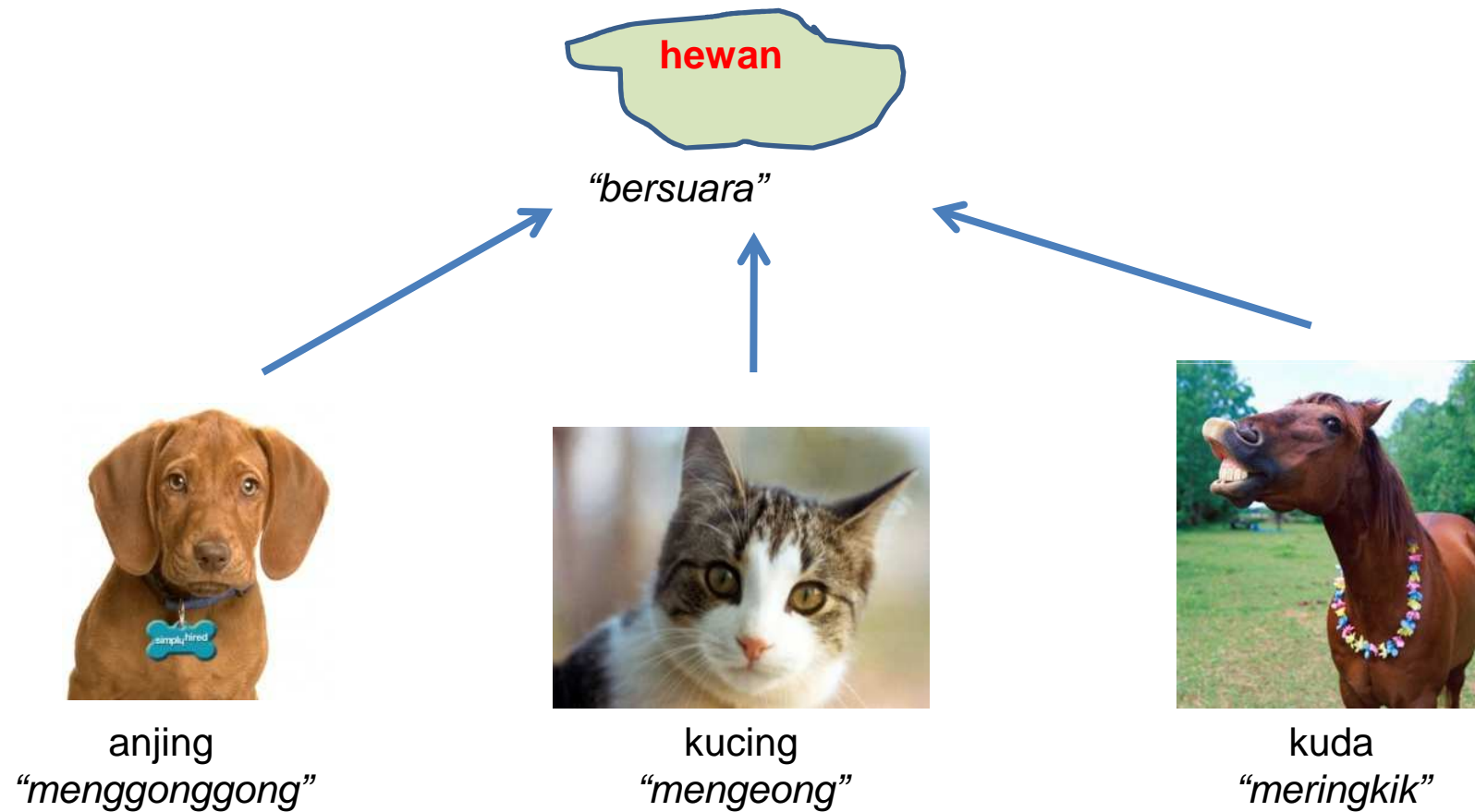
Penjelasan

- `A contoh = new B()`
- Maka obyek contoh sebenarnya bertipe class A namun berperilaku secara class B
- Atribut dan method pada obyek contoh akan mengikuti class A, bukan class B

Review: Overriding

- Subclass yang berusaha **memodifikasi** tingkah laku yang diwarisi dari superclass.
- **Tujuan:** subclass memiliki tingkah laku yang lebih **spesifik**.
- Dilakukan dengan cara *mendeklarasikan kembali* method milik parent class di **subclass**.

Polimorfisme - Overriding



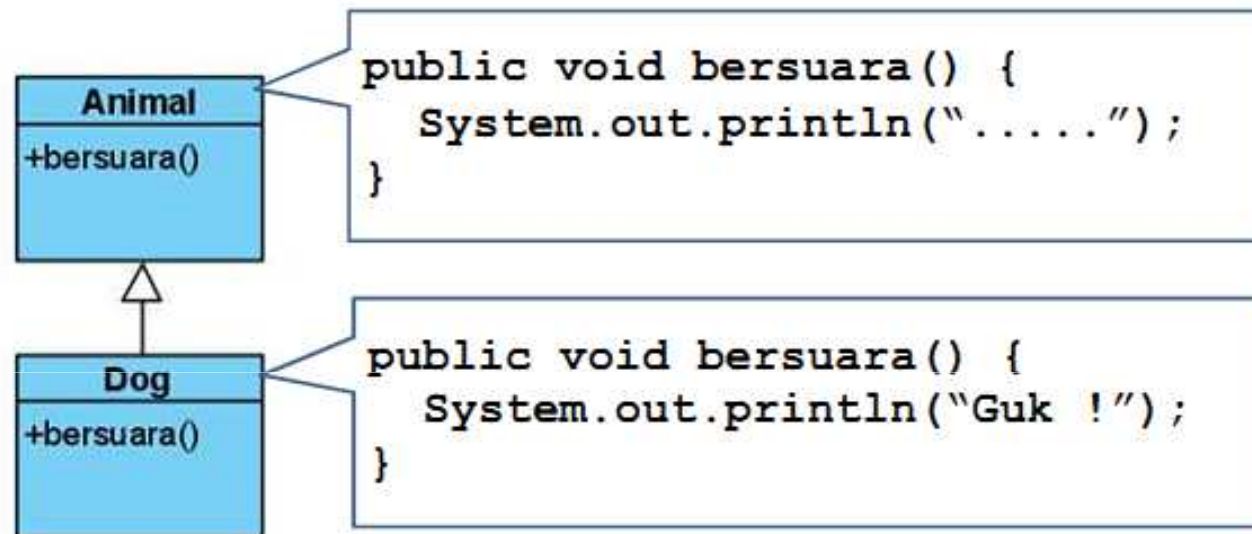
Review: Overriding

- Contoh:

```
class Animal {  
    public String bersuara() {  
        return "Suara binatang";  
    }  
}
```

```
class Dog extends Binatang {  
    public String bersuara() {  
        return "Guk guk";  
    }  
}
```

Penerapan

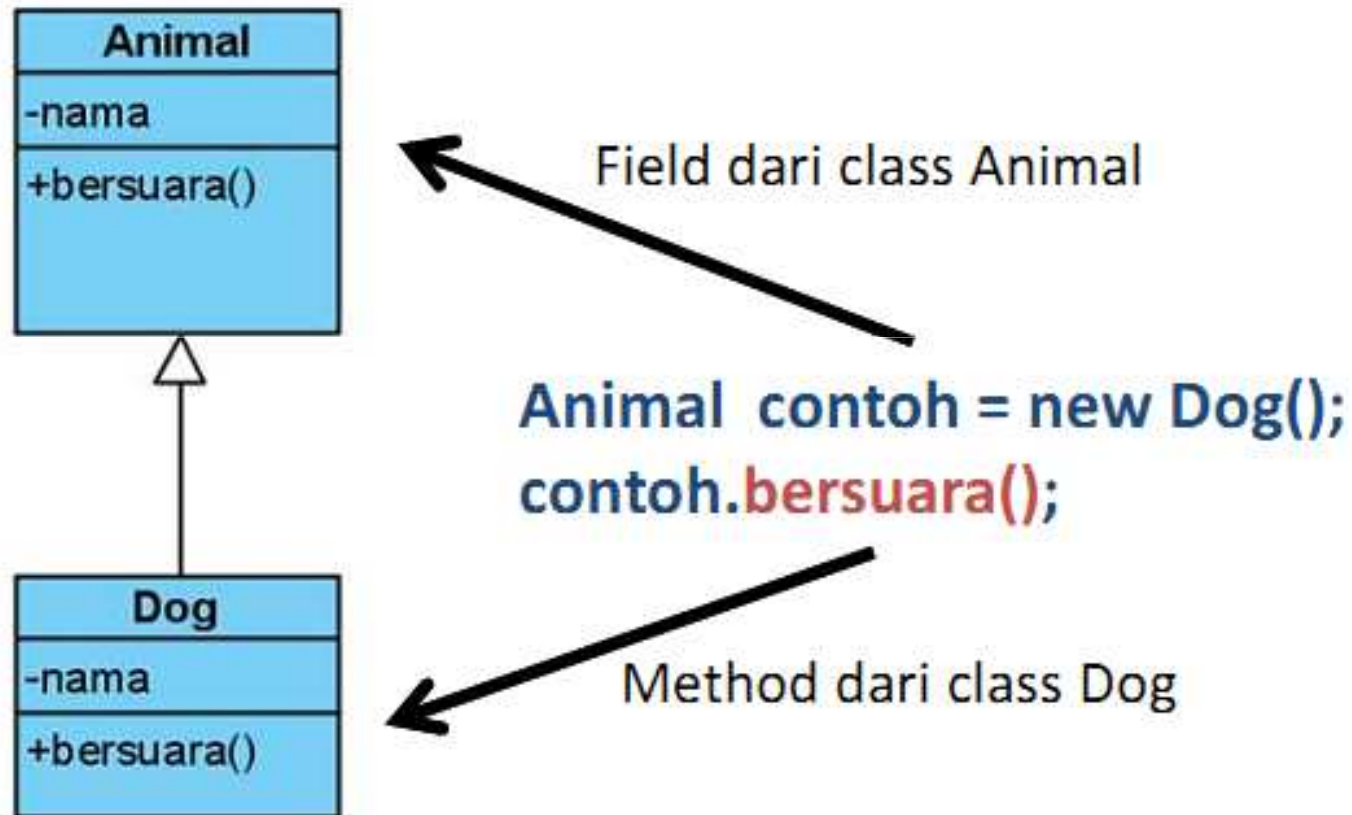


```
Animal contoh = new Animal();
contoh.bersuara();
```

```
Dog contoh = new Dog();
contoh.bersuara();
```

```
Animal contoh = new Dog();
contoh.bersuara();
```

Animal menjadi Dog



Review - Overloading

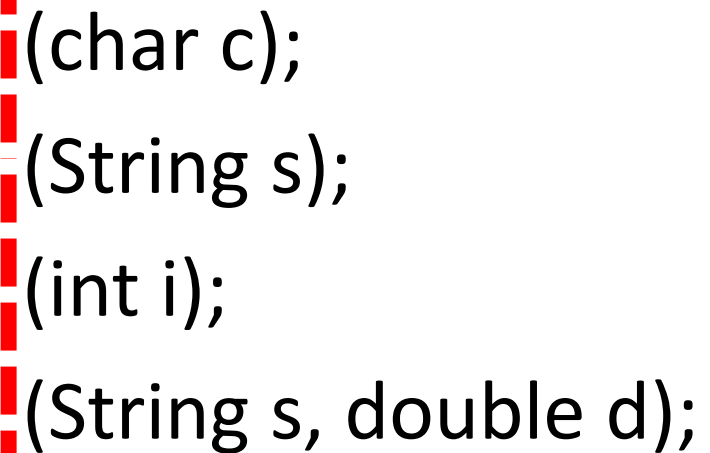
Aturan overloading:

- Nama method harus sama
- Daftar parameter harus berbeda
- Return type boleh sama, juga boleh berbeda

Overloading Method

- Contoh:

```
public void println(char c);  
public void println(String s);  
public void println(int i);  
public void println(String s, double d);
```



Contoh

```
class Induk{
    public void metode1(int x){
        System.out.println("Induk1");
    }
    public void metode1(int x,int y){
        System.out.println("Induk1-2");
    }
    public void metode2(String x){
        System.out.println("Induk2");
    }
}
```

```
D:\>java Anak
ini dari Anak1 1
Induk1-2
ini Anak 3 parameter
Induk1-2
Induk2
Induk2
MetodeAnak
ini dari Anak1 1
Induk1-2
Induk2
```

```
public class Anak extends Induk{
    public void metode1(int x){
        System.out.println("ini dari Anak1 "+x);
    }
    public void metode1(int x,int y,int z){
        super.metode1(x,y);
        System.out.println("ini Anak 3 parameter");
    }
    public void metodeAnak(){
        super.metode2("anton");
        System.out.println("MetodeAnak");
    }

    public static void main(String args[]){
        Anak a = new Anak();
        a.metode1(1);
        a.metode1(1,2,3);
        a.metode1(1,2);
        a.metode2("aaa");
        a.metodeAnak();

        Induk i = new Anak(); //polimorfisme
        i.metode1(1);
        i.metode1(1,2);
        i.metode2("aaa");
        //i.metodeAnak(); //error!
    }
}
```

Polimorfisme

- Berhubungan dengan **standarisasi**
- Pada hubungan **pewarisan**, class anak dan induk adalah sejenis
- Sifat “**sejenis**” dengan induknya itulah yang dapat digunakan untuk membuat kelas “**umum**” yang dapat menggunakan kelas yang bertipe induk namun bertindak laku sesuai dengan kelas anaknya

Class Bentuk

```
public abstract class Bentuk {  
    private String jenis;  
    public abstract void hitungLuas();  
    public void setJenis(String j){  
        this.jenis = j;  
    }  
    public String getJenis() {  
        return this.jenis;  
    }  
}
```

Class Segitiga


```
class Segitiga extends Bentuk{  
    private int alas;  
    private int tinggi;  
  
    public void hitungLuas() {  
        System.out.println ("Luas : "+(0.5*alas*tinggi));  
    }  
    public Segitiga(String j,int a,int t){  
        super.setJenis(j);  
        this.alas = a;  
        this.tinggi = t;  
    }  
}
```

Class Persegi

```
class Persegi extends Bentuk{  
    private int sisi;  
    public void hitungLuas() {  
        System.out.println ("Luas : "+(sisi*sisi));  
    }  
    public Persegi(String j,int s){  
        super.setJenis(j);  
        this.sisi = s;  
    }  
}
```

Class Pemakai

```
class Pemakai{  
    public void Laporkan(Bentuk b){  
        System.out.println ("Jenis : "+b.getJenis());  
        b.hitungLuas();  
    }  
  
    public static void main(String args[]) {  
        Pemakai p = new Pemakai();  
        Segitiga s = new Segitiga("segitiga",5,4);  
        Persegi pp = new Persegi("bujur sangkar",9);  
        p.Laporkan(s);  
        p.Laporkan(pp);  
    }  
}
```



```
C:\Program Files\Xinox Software\JCreatorV...  
Jenis : segitiga  
Luas : 10.0  
Jenis : bujur sangkar  
Luas : 81  
Press any key to continue...
```

Proses binding obyek oleh compiler

- Binding = pengikatan variabel obyek oleh compiler pada proses eksekusi program
- Ada 2 jenis:
 - Static binding
 - Dyanamic binding

Static Binding

- Binding jenis ini dilakukan saat **compile-time**
- Disebut juga **early binding**
- Contoh: pada “Overloading”, yang berarti ada beberapa class method dengan nama yang sama
 - Method yang **dijalankan** tergantung dari parameter yang diberikan
- Pemilihan method yang dijalankan ditentukan saat **compile time** (static binding)

Static Binding

Contoh:

```
public void println (char c);  
public void println (String s);  
public void println (int i);  
public void println (String s, double d);
```

```
println("PBO");  
println(2000);  
println("a");  
println('b');  
println("Data", 2.0);
```

Dynamic Binding

- Binding jenis ini dilakukan saat **run-time** (dynamic)
- Sering disebut dengan **Late Binding**
- Penentuan method mana yang dijalankan dilakukan saat **run-time**
- Dynamic binding dapat dilakukan dengan 2 cara:
 - Overriding Inheritance Method
 - Overriding Interface Method

Dynamic Binding

Contoh:

```
void kill (Mortal m) {  
    m.killed();  
}
```

- Misalnya **orang** dan **tanaman** masuk dalam class **Mortal**, maka method **killed()** yang dipanggil tergantung dari siapa pemanggilnya.

Contoh

```
class Mortal{
    public void showMortal(){ system.out.println("Mortal"); }
    public void killed(){ }
}

class Tree extends Mortal{
    public void showTree() { system.out.println("Tree"); }
    public void killed() { system.out.println("Tree killed"); }
}

class Animal extends Mortal{
    public void showAnimal() { system.out.println("Animal"); }
    public void killed() { system.out.println("Animal killed"); }
}
```

```
E:\Documents\Dosen\pbo\latihan>java Utama
Tree
Animal
Animal killed
Tree killed
```

```
public class Utama{
    public void Kill(Mortal m){
        m.killed();
    }
    public static void main(String args[]){
        Tree t = new Tree();
        t.showTree();
        Animal a = new Animal();
        a.showAnimal();
        Utama u = new Utama();

        u.Kill(a);
        u.Kill(t);

    }
}
```

Polimorfisme dengan Interface

- Suatu interface dapat mengekstens interface lain
- Tujuannya: memperluas kemampuan interface
- Interface hanya bisa extends interface saja
 - Tidak bisa extends class & abstract class

Overriding Interface Method

```
class A extends Object{  
    public String toString(){  
        return "toString in A";  
    }  
    public String x(){  
        return "x in A";  
    }  
}
```



Method toString() berasal dari class Object

Overriding Interface Method

Contoh:

```
interface I1{  
    public void p();  
}
```

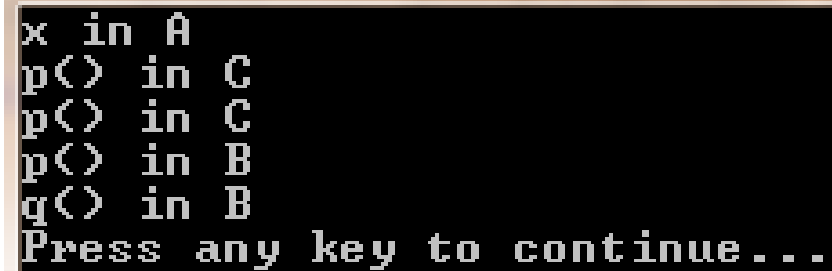
```
interface I2 extends I1{  
    public void q();  
}
```

Overriding Interface Method

```
class B extends A implements I2{  
    public void p(){ System.out.println ("p() in B"); }  
    public void q(){ System.out.println ("q() in B"); }  
}
```

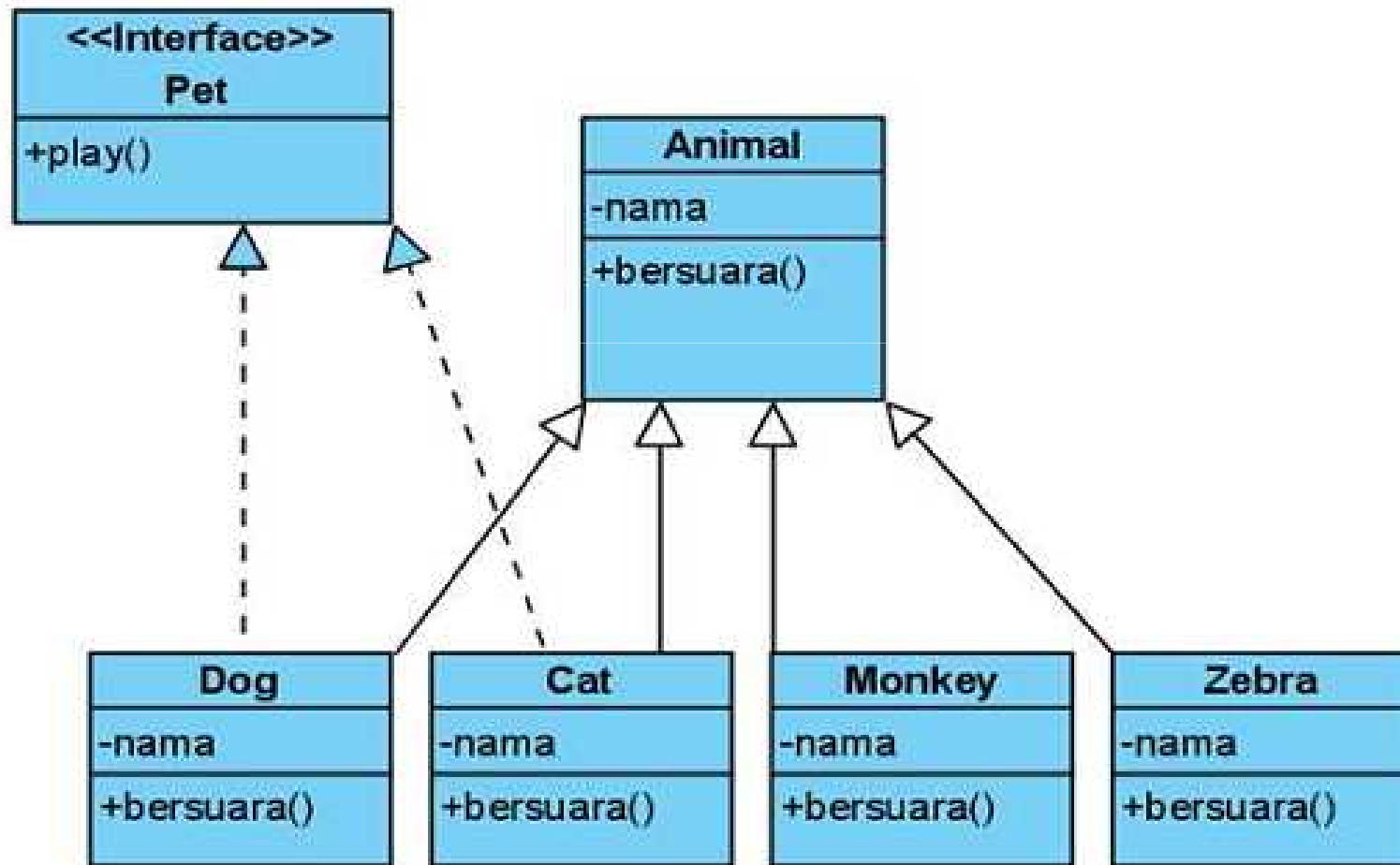
```
class C extends Object implements I2{  
    public void p(){ System.out.println ("p() in C"); }  
    public void q(){ System.out.println ("p() in C"); }  
}
```

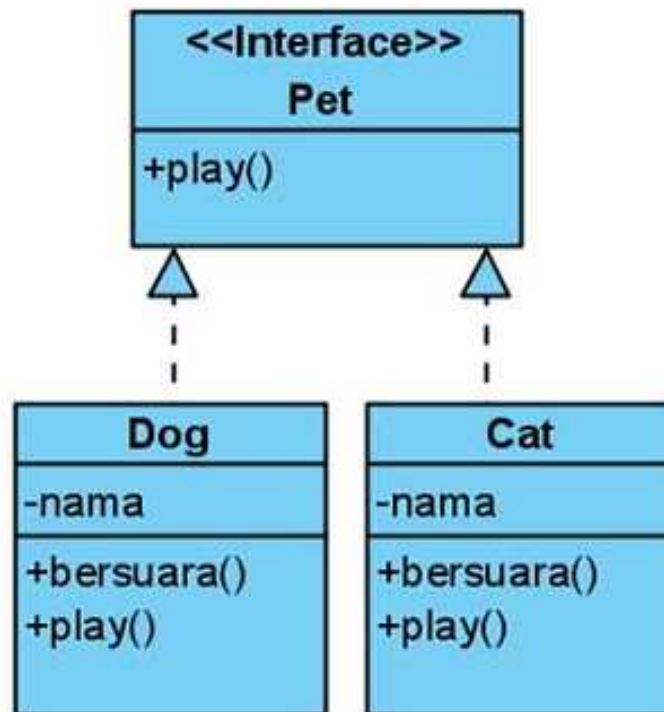
```
public class demoInterface {  
    public static void main(String[] args) {  
  
        A a = new A();  
        System.out.println (a.x());  
  
        C c = new C();  
        c.p();  
        c.q();  
  
        B b = new B();  
        b.p();  
        b.q();  
  
    }  
}
```



```
x in A  
p() in C  
p() in C  
p() in B  
q() in B  
Press any key to continue...
```

Contoh lain interface





Class Dog dan Cat harus membuat implementasi method **play()** dari interface **Pet**

Kode

```
class Dog extends Animal implements Pet {  
    ...  
    public void play() {  
        System.out.println("Dog is playing");  
    }  
}
```

```
class Cat extends Animal implements Pet {  
    ...  
    public void play() {  
        System.out.println("Cat is playing");  
        System.out.println("Cat is sleeping");  
    }  
}
```

Hasil

```
Pet[] petList = new Pet[5];  
petList[0] = new Dog("husky");  
petList[1] = new Cat("felix");  
petList[2] = new Dog("waldo");  
petList[3] = new Cat("garry");  
petList[4] = new Dog("froddo");
```

```
petList[0].play();  
petList[3].play();
```

petList[0].bersuara() ?

NEXT

- Package & Relasi antar Class